Chapter 33
Optimal Link Path Queries in a Simple Polygon

Esther M. Arkin* Joseph S. B. Mitchell† Subhash Suri‡

Abstract

We develop a data structure for answering link distance queries between two arbitrary points in a simple polygon. The data structure requires $O(n^3)$ time and space for its construction and answers link distance queries in $O(\log n)$ time. Our result extends to link distance queries between pairs of segments or polygons. We also propose a simpler data structure for computing a link distance approximately, where the error is bounded by a small additive constant. Finally, we also present a scheme for approximating the link and the shortest path distance simultaneously.

1 Introduction

Motivation. The study of link metric is motivated by applications in several areas of computer science, such as robotics, motion planning, VLSI, computer vision, and solid modeling; see for instance [9, 11, 15, 17, 19]. In an environment cluttered with polygonal obstacles, the *link distance* between two points s and t is the minimum number of edges in any polygonal path between them, avoiding the interiors of all the obstacles. In many problems, the link distance provides a more natural measure of path complexity than the standard Euclidean distance. In complex robot systems, for instance, a straight-line navigation is often considerably cheaper than rotation and, therefore, minimizing the number of turns is a natural objective in path planning [13, 15, 19].

In graph-layout problems, it is often desirable to minimize the number of bends [16, 21]. The same problem also arises in wiring a circuit board. Although a circuit board is not a simple polygon, the designers often specify the "homotopy" class of a wiring path, which reduces the problem to a minimum link path inside an appropriate simple polygon [4].

The minimum link paths are also used for curve-compression in solid modeling [12]. A typical problem in solid modeling is to intersect surfaces. The intersection of two surfaces can be quite complex even if the surfaces themselves are of relatively low degree. To avoid the combinatorial explosion inherent in the closed form solution, it is convenient to compute a piecewise linear approximation to the curve of intersections. Natarajan proposes to replace an intersection curve A by its minimum link analog, which is computed inside an ε-fattening of A [12]. Related work on approximating simple polygons and polygonal maps is discussed in [6].

Our results. The main result of our paper is a data structure for answering link distance queries between two *arbitrary* points inside a simple polygon. The data structure requires $O(n^3)$ preprocessing time for its construction, and answers a query in $O(\log n)$ time, where n denotes the number of vertices in the polygon. An easy lower bound argument shows that this is the best possible query time, even if all the queries are asked with respect to a fixed source point. Our result extends to the more general case of segments or polygons; the link distance between two polygons S and T is the minimum number of links needed to join any point of S to any point of T. If S and T are convex polygons contained in P, having a total of k vertices, we can compute their link distance in $O(\log k \log n)$ time; thus, a query between two line segments takes $O(\log n)$ time. If S and T are nonconvex but simple polygons, with a to-

*Dept. of Applied Math, SUNY, Stony Brook, NY 11794-3600. Partially supported by NSF Grants DMS-8903304 and ECSE-8857642.

†Dept. of Applied Math, SUNY, Stony Brook, NY 11794-3600. Partially supported by a grant from Hughes Research Laboratories, Malibu, CA, and by NSF Grant ECSE-8857642.

‡Bell Communications Research, Room 2Q-358, 445 South Street, Morristown, NJ 07960

tal of k vertices, then one cannot achieve a query time that is polylogarithmic in both k and n. We establish a lower bound of $\Omega(k+\log n)$, and we present an algorithm with running time $O(k \log^2 n)$ time. In each of the above cases, after processing a link distance query, we can output a minimum-link path between the two query objects in time proportional to the link distance.

The algorithm for computing the exact link distance is quite complex and requires a relatively high $O(n^3)$ preprocessing time. If, however, one is willing to accept a path with at most one extra link, then a simpler algorithm, based on $O(n^2)$ time and space preprocessing, is also described. Note that it is straightforward to preprocess a polygon (even with holes) in order to support approximate link distance queries between two points, as follows. We precompute the link distances between all pairs of vertices. Then, given two query points s and t, we report their link distance as 2 plus the link distance between s' and t', where s' and t' are two vertices of P, respectively, visible from s and t. The visible vertices s' and t' can be found in $O(\log n)$ time by point location in a triangulation of P. It is easy to see that this approximation yields an (additive) error of at most four links. This scheme yields an $O(n^2)$ time and space data structure that supports approximate link distance queries in $O(\log n)$ time. The challenge in link distance query problems is to obtain *exact* answers (or, to reduce the approximation error below 4).

Finally, we consider the problem of simultaneously minimizing the number of links and the length of a path inside a simple polygon. We show that for any two points s and t, there exists a path that uses at most twice the minimum number of links and whose length is at most $\sqrt{2}$ times the shortest path from s to t.

Relation with previous work. The problem of finding a minimum link path between two *fixed* points inside a simple polygon was solved in [17]. The algorithm runs in linear time, provided that the polygon is triangulated. (A recent result of Chazelle [3] gives a linear time triangulation algorithm.) In [19], a more general framework was established for link distance, based on *window partition*. Using window partition trees, the link distance queries from a *fixed* source can be answered in logarithmic time, after a linear time preprocessing. Efficient algorithms for link diameter and link center are given in [8, 9, 18]. Finally, the problem of computing a minimum link path between two fixed points in a multiply connected polygon was considered in [11]. Their algorithm runs in time $O(n^2 \log^2 n)$.

An optimal algorithm for computing the length of the shortest path between two arbitrary points in a simple polygon was developed by Guibas and Hershberger [5]. Our algorithm resolves the same problem for the link metric, although at a higher preprocessing cost. The increased preprocessing is partly attributable to a key difference between a shortest path and a minimum link path. While the shortest path in a simple polygon is unique and turns only at the vertices of the polygon, a link path is far from unique and can turn at any point inside the polygon. Thus, to preprocess the polygon for link distance queries, we need to build and search a much larger graph than what is needed for a shortest path query. In partial compensation, though, the same preprocessing also enables us to compute link distances between more complex objects, such as line segments and convex polygons, at little additional cost.

If the polygon is *rectilinear*, a recent result of de Berg [2] shows how to preprocess it in $O(n \log n)$ time and space so as to compute *rectilinear* link distance between two arbitrary query points in $O(\log n)$ time.

Organization of paper. In Section 2, we give definitions and notation. Section 3 describes our algorithm for the two-point link distance query, and Section 4 generalizes it to segment and polygon queries. Section 5 describes a method for simultaneously approximating the minimum link and geodesic path. Section 6 concludes with open problems.

2 Preliminaries

Let P be a simple polygon in the plane on n vertices. Given two points s and t in P, a *minimum-link path* between them is a polygonal path inside P with a minimum number of edges, also called links. Figure 1 shows an example. The *link distance* $d_L(s,t)$ refers to the number of links in a minimum-link path between s and t.

The following notion of a window partition, introduced in [19], is central to our discussion. Given a point or a line segment s, let $WP(s)$ denote the subdivision of the polygon P into maximally-connected regions with the same link distance from s. We call $WP(s)$ the *window partition* of P with respect to s. A window partition of P can be computed in $O(n)$ time. $WP(s)$ has an associated set of *windows*, which are chords of the polygon that serve as boundaries between adjacent regions of the partition. We can preprocess a window partition $WP(s)$, in additional $O(n)$ time, for point location queries, after which a link distance query from the fixed source s can be answered in time $O(\log n)$. (A minimum-link path from s to the query point t can be traced in constant time per link.) An example of a window partition is shown in Figure 1.

Two points s and t in P are *mutually visible* if the line segment $\overline{st}$ does not intersect the exterior of P. The visibility graph of P, denoted $VG(P)$, is the graph on the vertex set of P whose edges join mutually visible pairs of vertices. Given a visibility graph edge $\overline{ab}$, the *extension* of $\overline{ab}$ refers to the connected component of $\ell \cap P$ that contains $\overline{ab}$, where ℓ is the line passing through a and b.

It is well-known that, for any two points s and t in P, the shortest (or, *geodesic*) path $\pi_G(s,t)$ is unique and turns only at the vertices of P. A minimum link path, on the other hand, is generally not unique. For the sake of consistency, we will always use the following uniquely defined representative path $\pi_L(s,t)$, which we call the *greedy* minimum link path. The path $\pi_L(s,t)$ uses only the extensions of the windows of $WP(s)$ and the last link is chosen to pass through the last vertex of the geodesic path $\pi_G(s,t)$. We need to define a few more concepts before we can describe our algorithm.

Given a polygonal path π, an interior edge $e \in \pi$ is called an *inflection edge* if the predecessor and the successor edges of e lie on opposite sides of the line containing e. An edge e is said to be (rotationally) *pinned* if it passes through two vertices u and v of P in such a way that e is locally tangent at u and v on opposite sides of e.

A window partition $WP(s)$ has a set of associated windows. A window $\overline{xy}$ is a line segment such that the endpoint x is a reflex vertex v of P and the endpoint y lies on an edge e of P — the endpoint x is closer to s by geodesic distance. The combinatorial type of window $\overline{xy}$ is the vertex-edge pair (v,e). The *combinatorial type* of a window partition $WP(s)$ is a listing of the combinatorial types of all of its windows.

3 Two-Point Queries

We begin by describing our algorithm for the case of link distance query between two points. We assume in the following that the polygon P is triangulated; this can be done in $O(n)$ time using a recent result of Chazelle [3]. A further linear-time preprocessing also prepares the polygon for point-location queries. Given two arbitrary points s and t, we can check in $O(\log n)$ time if either s or t lies outside P. If so, we halt; otherwise, we proceed to compute their link distance in P. We first describe the preprocessing phase of our algorithm, which builds the data structure used to answer the link distance query.

3.1 Preprocessing

We perform the following four preprocessing steps on the polygon P, each resulting in a data structure suited for a specific query.

(P1) Build a data structure for answering shortest path (geodesic) queries between two arbitrary points in P. Enhance the data structure so that it allows us to determine if there is an inflection edge in $\pi_G(s,t)$, and, if so, produce one.

(P2) Construct the visibility graph $VG(P)$.

(P3) Construct the window partitions $WP(v)$ for each vertex $v \in P$. Enhance these partitions so that, for any vertex $v \in P$ and an arbitrary query point t, we can find the last pinned edge along the greedy link path $\pi_L(v,t)$.

(P4) For each of $O(n^2)$ "atomic segments" $\overline{pq}$ on the boundary of P, compute $WP(\overline{pq})$. For each greedy path determined by $WP(\overline{pq})$ that has no pinned edges, compute and store "projection functions" $f_i(\cdot)$. The atomic segments and projection functions are described below.

We now elaborate on each of the above steps.

Step (P1). The data structure for two-point geodesic queries is built using the algorithm of Guibas and Hershberger [5]. Their data structure requires $O(n)$ time and space for construction, and answers a geodesic query in logarithmic time. In particular, given a pair of query points s and t in P, the length of the path $\pi_G(s,t)$ can be computed in $O(\log n)$ time. If desired, the path $\pi_G(s,t)$ can be listed at the expense of constant time per edge.

A minor enhancement of the data structure of Guibas and Hershberger allows us to report an inflection edge in the path $\pi_G(s,t)$. The algorithm in [5] builds a hierarchical data structure of "hourglasses", which are subpolygons defined by a pair of diagonals of the triangulation and the geodesic paths between their endpoints. In processing a query, an implicit representation of the shortest path $\pi_G(s,t)$ is obtained by concatenating $O(\log n)$ hourglasses. (The amortized cost of concatenating two hourglasses is constant.) We "remember" a representative inflection edge for each pre-stored hourglass. In addition to the pre-stored edges, the only other candidates for inflection edges are internal tangents between two adjacent hourglasses during the concatenation. These edges are computed as part of the concatenation, and there are $O(\log n)$ of them, all of which we can check.

Step (P2). We construct the visibility graph VG(P), and compute the extensions of all the visibility graph edges. This takes time proportional to the size of the visibility graph, by a result due to Hershberger [7]. The algorithm in [7], in fact, outputs the visibility graph edges in sorted order about each vertex. Thus, we can easily answer the following ***ray shooting query*** in logarithmic time: given a vertex v and a direction θ, find the first point p where the ray from v in direction θ exits the interior of P.

Step (P3). We construct the window partitions $WP(v)$ for all vertices v of P, and preprocess each partition for efficient point location queries. This data structure allows one to compute the link distance between any vertex $v \in P$ and an arbitrary point t in $O(\log n)$ time. Since a single window partition can be constructed and preprocessed in linear time [19], this step requires altogether $O(n^2)$ time and space.

We need to maintain one additional piece of information with each cell of the window partition $WP(v)$: the last pinned edge, if any, along the greedy path from v to points within the cell. It is a simple matter to maintain this pinned edge information as the window partition is constructed.

The endpoints of the windows in all of these partitions lie on the boundary of P, and together with the vertices of P they partition the boundary of P into $O(n^2)$ intervals, which we call *atomic segments.* In $O(n^2 \log n)$ time we sort the endpoints of windows along boundary edges of P, so that we have access to an ordered list of atomic segments on each edge of P. The relevance of atomic segments is stated in the following lemma.

Lemma 3.1 *If $\overline{pq}$ is a atomic segment on the boundary of P, then the combinatorial type of the window partition $WP(z)$ is the same for all points z in the interior of the segment $\overline{pq}$.*

Proof. The combinatorial type of $WP(z)$ changes only when some window of $WP(z)$ overlaps a visibility graph edge. Assume that $z' \in \overline{pq}$ is a point corresponding to such an event, and let $\overline{uv}$ be the corresponding visibility graph edge, where v is further from z' than u. Refer to Figure 2. Then, z' must be the endpoint of a window in the partition $WP(v)$, implying that z' can only occur as an endpoint of a atomic segment. □

Step (P4). For each of the $O(n^2)$ atomic segments, we construct the window partition $WP(\overline{pq})$. This takes $O(n^3)$ total time.

During the construction of $WP(\overline{pq})$, we keep track of the following additional information. Let $\overline{x_i y_i}$ be a window of $WP(\overline{pq})$, with x_i closer to $\overline{pq}$, such that that the greedy link path from any point $z \in \overline{pq}$ to x_i has no pinned edge. Let e_i be the edge of P containing point y_i. As z varies along atomic segment $\overline{pq}$, the window endpoint y_i varies along edge e_i according to a *projection function* $f_i(z)$, which can be written as a fractional linear form:

$$y_i(z) = f_i(z) = \frac{A_i + B_i z}{C_i + D_i z},$$

with the constants A_i, B_i, C_i, D_i obtainable from $A_{i-1}, B_{i-1}, C_{i-1}, D_{i-1}$ in constant time. (These functions were introduced in [1] in the context of nested polygons, and the reader is referred to [1, 20] for details regarding these functions.)

We compute and store the functions f_i that correspond to all windows whose greedy paths from $\overline{pq}$ contains no pinned edge. With this preprocessing, we can evaluate the exact position of y_i corresponding to an arbitrary point $z \in \overline{pq}$ (or vice-versa) in constant time.

Summary All of the above preprocessing requires altogether $O(n^3)$ time and space. The most expensive step is (P4); others require $O(n^2 \log n)$ time and $O(n^2)$ space.

3.2 Processing a Query

In this section, we show how the preprocessing of the preceding section is used to answer the link distance query between two arbitrary points s and t. Our query processing algorithm proceeds as follows:

(Q1) We compute an implicit representation of the shortest path $\pi_G(s,t)$, and extract from it the first vertex v and the last vertex w. By the data structure of Step (P1), this requires $O(\log n)$ time.

(Q2) If neither v nor w exists, then it is easily seen that the link distance $d_L(s,t) = 1$. If $v = w$, then $d_L(s,t) = 2$. Otherwise, $v \neq w$, and we determine if there is an inflection edge in the path $\pi_G(s,t)$; this also takes logarithmic time. We go to Step (Q3) if $\pi_G(s,t)$ has an inflection edge, and to Step (Q4) otherwise.

(Q3) (*Path $\pi_G(s,t)$ has an inflection edge $\overline{ab}$.*) Assume that b precedes a on the path $\pi_G(s,t)$. (Refer to Figure 3.) Then, there exists a minimum-link path between s and t one of whose links contains $\overline{ab}$. Using the data structure of Step (P3), we locate s in $WP(a)$ and t in $WP(b)$, and compute the link distances $d_L(a,s)$ and $d_L(b,t)$ in $O(\log n)$ time. The link distance between s and t is $d_L(s,t) = d_L(s,a) + d_L(b,t) - 1$.

(Q4) (*Path $\pi_G(s,t)$ has no inflection edge.*) Backproject $\overline{sv}$ to obtain the point s' where the ray from v in the direction of s exits the polygon P. The visibility graph data structure of Step (P2) allows us to do this in $O(\log n)$ time. Notice that $d_L(s',t) = d_L(s,t)$.

We find the atomic segment $\overline{pq}$ containing the point s' on the boundary of P; this takes $O(\log n)$ time by binary search on the sorted list of atomic segments. We locate t in $WP(\overline{pq})$, and determine if there exists a pinned edge in the greedy path from $\overline{pq}$ to t. We go to Step (Q4.1) if there exists a pinned edge, and to Step (Q4.2) otherwise.

(Q4.1) (*There is a pinned edge $\overline{ab}$ in the greedy path from $\overline{pq}$ to t.*)

Assume that b precedes a on the path $\pi_L(\overline{pq},t)$. (Refer to Figure 4.) Locate t in $WP(b)$ and s in $WP(c)$, where c is the endpoint of the extension of $\overline{ab}$ closer to a. Then, $d_L(s,t) = d_L(s,c) + d_L(b,t) - 1$.

(Q4.2) (*No pinned edge in the greedy path from $\overline{pq}$ to t.*)

Let b denote the last bend point in the path $\pi_L(\overline{pq},t)$, and let e be the edge of P that contains

b. Let r be the "pivot" vertex of the polygon P incident on the link to b. Refer to Figure 5. We evaluate the function $f(s')$, where f corresponds to the (combinatorial type of the) path from $\overline{pq}$ to e. Let $f(s') = a$, where a is a point on the edge e. We backproject $\overline{tw}$ to obtain a point w' where the ray from t in the direction of w exists the interior of P; recall that w is the last vertex on the geodesic path between s and t. If $\overline{tw'}$ intersects $\overline{ra}$, then $d_L(s,t) = d_L(\overline{pq},t)$; otherwise, $d_L(s,t) = d_L(\overline{pq},t)+1$.

The following lemma shows that the link distance is computed correctly in Step (Q4.2).

Lemma 3.2 *If $\overline{tw'}$ intersects $\overline{ra}$, then $d_L(s,t) = d_L(\overline{pq},t)$, otherwise $d_L(s,t) = d_L(\overline{pq},t)+1$.*

Proof. For any point x on the segment $\overline{pq}$, the following inequality clearly holds:

$$d_L(\overline{pq},t) \leq d_L(x,t) \leq d_L(\overline{pq},t)+1. \qquad (1)$$

Since b is the last bend point in $\pi_L(\overline{pq},t)$ and since $\pi_L(s,a)$ has the same combinatorial type as $\pi_L(\overline{pq},b)$, we must have $d_L(s,a) = d_L(\overline{pq},b) = d_L(\overline{pq},t)-1$.

Now if the segment $\overline{tw'}$ intersects $\overline{ra}$, we get

$$d_L(s,t) = d_L(s,a)+1 = d_L(\overline{pq},t).$$

Otherwise, since b and t are mutually visible and b and a are mutually visible, we get

$$d_L(s,t) = d_L(s,a)+2 = d_L(\overline{pq},t)+1.$$

This completes the proof. □

This completes the description of our query algorithm. Using the data structures built in the preprocessing phase, each step of the algorithm takes $O(\log n)$ time. The following theorem summarizes this result.

Theorem 3.3 *Given a simple polygon of n vertices, we can preprocess it in time and space $O(n^3)$ so that, for any two points s and t, we can find their link distance $d_L(s,t)$ in $O(\log n)$ time.*

Given the relatively high (cubic) cost of our data structure, it is reasonable to ask if one can get *close* to the correct link distance with less time and space. If one is willing to tolerate an error of ± 2, or a one-sided error of at most 4 links, then the preprocessing expense can be reduced to $O(n)$, while retaining the same query performance, see [19].

Our next result reduces the approximation error from 4 to 1, but at the expense of increased preprocessing.

Theorem 3.4 *Given a simple polygon of n vertices, we can preprocess it in time and space $O(n^2)$ so that, for any two points s and t, we can find an approximate link distance $\tilde{d}(s,t)$ in $O(\log n)$ time such that $d_L(s,t) \leq \tilde{d}(s,t) \leq d_L(s,t)+1$.*

Proof. The polygon is preprocessed using only the first three steps, namely, Steps P1–P3. This takes $O(n^2)$ time and space.

A query is processed as follows. Find the first vertex v on the geodesic path from s to t. If no such vertex exists, we know that $d_L(s,t) = 1$, so set $\tilde{d}(s,t) = 1$. Otherwise set $\tilde{d}(s,t) = d_L(v,t)+1$. By the preprocessing step, v and $d_L(v,t)$ can be found in $O(\log n)$ time.

Clearly, $d_L(s,t) \leq \tilde{d}(s,t)$, since the edge $\overline{sv}$ together with $\pi_L(v,t)$ gives a path of $\tilde{d}(s,t)$ links. It remains to be shown that $\tilde{d}(s,t) \leq d_L(s,t)+1$, or in other words, that $d_L(v,t) \leq d_L(s,t)$. To establish that, we extend a ray from s through v until it first exits the polygon. This extension partitions the polygon into two subpolygons, one containing s and the other containing t. (If s and t were in the same subpolygon, then the geodesic path from s to t would also lie completely in the same subpolygon and would make a convex turn at vertex v, which is impossible.) Now, consider a minimum link path from t to s. It must cross the extension of $\overline{sv}$, at which point v becomes visible to the path, yielding a path from t to v with $d_L(s,t)$ links. This shows that $d_L(v,t) \leq d_L(s,t)$, and the proof is complete. □

3.3 Remarks

Interestingly enough, just as in the minimum-link nested polygon problem [1,22], the nonconvex case turns out to be *easier*: If the (greedy) minimum-link path from s to t is "nonconvex" (i.e., there is an inflection edge in the path), then $O(n^2)$ time and space suffice. The hard case turns out to be when the path is constant-turning, and is free to rotate. The difficulty comes in deciding the last link, and this is the only part of the algorithm that requires cubic preprocessing.

If we are interested in minimum-link *rectilinear* paths from s to t within a rectilinear polygon, then we can modify our methodology to give a relatively simple $O(n^2)$ preprocessing algorithm that allows for $O(\log n)$ time (exact) queries. Through each vertex v, we construct a maximal horizontal and a maximal vertical segment within P. The endpoints of these segments, together with the vertices of P, partition the boundary of P into $O(n)$ pieces that can serve as atomic segments in our method. Recently, de Berg [2] has improved this result by giving an algorithm whose preprocessing time and space is $O(n \log n)$, with optimal $O(\log n)$ query time.

4 Generalizations

In this section, we extend our results to more general objects. We first consider the case where s and t are line segments, and then generalize to polygons.

4.1 Query between two segments

Let $\sigma = \overline{ss'}$ and $\tau = \overline{tt'}$ be two line segments in P. We want to compute the link distance $d_L(\sigma, \tau)$. We perform the same preprocessing steps P1–P4 as in Section 3.1. (With this preprocessing, we can also check in $O(\log n)$ time if indeed $\sigma, \tau \subset P$.)

To process a link distance query between segments, we essentially reduce the problem to several link distance query problems between pairs of points. We need the following fact.

Lemma 4.1 *Given the window partition $WP(s)$, the link distance from s to any line segment $\tau \subset P$ can be determined in time $O(\log n)$.*

Proof. Locate one endpoint of τ in $WP(s)$, and then "walk" along τ to its other endpoint, by performing $O(\log n)$-time ray-shooting queries. For each cell of the window partition that is crossed, find the link distance of that cell from s, by checking the label of that cell. Clearly, the link distance $d_L(s, \tau)$ is the minimum label encountered during the walk. We show below that a segment can cross at most a constant number (three) of cells of a window partition, and hence the above procedure can be carried out in $O(\log n)$ time.

It is easy to see that a segment cannot intersect two cells of a window partition whose link distance to the source differ by more than one (otherwise the definition of a window partition is violated). Now, if a segment τ crosses four or more cells, there must be three consecutive cells whose labels are k, $k+1$, and k, for some $k \geq 1$. The following argument shows that this cannot occur.

Consider the graph-theoretic dual of the window partition $WP(s)$. This is a tree, called a *window tree*, see [18,19] for details. The window tree has a node for each cell of $WP(s)$ and two nodes are connected by a tree edge if the cells share a boundary edge. The root corresponds to the cell containing s, and a node at depth i corresponds to a cell whose points have link distance $i+1$ from s.

If the segment τ intersected three nodes, of depth $k-1$, k, and $k-1$, then there must be an edge between the nodes corresponding to the labels $k-1$. But since the window tree is acyclic, there cannot be an edge between two nodes of the same depth. Thus, τ can intersect at most three cells, with depths k, $k-1$, and k. This completes the proof. □

Answering the Query

Our query algorithm performs the following steps.

(Q1) Compute an implicit representation of each of the geodesic paths $\pi_G(s,t)$, $\pi_G(s,t')$, $\pi_G(s',t)$, $\pi_G(s',t')$. Determine if any of these four paths contains an inflection edge; using our preprocessing, this can be accomplished in $O(\log n)$ time. If there are no inflection edges, we go to Step (Q2), otherwise we go to Step (Q3).

(Q2) (*Each of the four paths is inflection-free.*) There are only two possibilities: Either the four paths form an open hourglass in which at least one pair of paths is disjoint (Figure 6, or they form a closed hourglass and every pairs of paths intersects (Figure 7). In order to distinguish between the two, we determine if the four paths are pairwise edge-disjoint. Since all the paths are inflection-free, it suffices to examine only the first edge of each of the two paths incident to each endpoint, which can be done in constant time, given the implicit representation of the paths. We have the following two subcases.

(Q2.1) (*At least one pair of disjoint paths.*) Fig. 6.

In this case, the two paths incident to one of the endpoints must be edge-disjoint, and we output $d_L(\sigma, \tau) = 1$.

(Q2.2) (*Non-disjoint paths.*) Fig. 7.

All four paths share a common (inflection-free) subpath $\pi_G(u,v)$, where the vertices u and v form the apexes of the "funnels" based on the segments σ and τ. In this case, we output $d_L(\sigma, \tau) = \min\{d_L(s,t), d_L(s,t'), d_L(s',t), d_L(s',t')\}$.

(Q3) (*At least one path contains an inflection edge.*) Let the inflection edge be $\overline{uv}$, with vertex u being closer to σ. Refer to Figure 8. In this case, we output $d_L(\sigma, \tau) = d_L(u, \tau) + d_L(v, \sigma) - 1$.

By Lemma 4.1, each of the link distances required in the processing of a query can be computed in $O(\log n)$ time. We now show that in each case the link distance was computed correctly.

Lemma 4.2 *[Case (Q2.2)] If the four geodesic paths are inflection-free and pairwise non-disjoint, then $d_L(\sigma, \tau) = \min\{d_L(s,t), d_L(s,t'), d_L(s',t), d_L(s',t')\}$.*

Proof. We recall that the four paths share a common (inflection-free) subpath $\pi_G(u,v)$, where the vertices u and v form the apexes of the "funnels" based on the segments σ and τ. We consider one of the four paths to be the *outer* path and one to be the *inner* path. In Figure 7, $\pi_G(s,t)$ is the outer and $\pi_G(s',t')$ is the inner path. (The outer and inner paths can be determined in constant time simply by examining the nature of the geodesic paths at their endpoints.) The portions of the outer path between u and s, and between v and t, consist of a single edge. This must be the case because no vertex of the polygon can cause a bend in a geodesic path between u and s, while keeping the path between s and t inflection free.

Thus, we have the following inequality:

$$d_L(\sigma,\tau) \le \min\{d_L(s,t), d_L(s,t'), d_L(s',t), d_L(s',t')\}.$$

To complete the proof, we show that the $d_L(\sigma,\tau)$ equals the link distance between the endpoints of the outer path, namely, s and t.

Let $x \in \sigma$, where $x \neq s$, be a point yielding a minimum link path to τ, shown by the dotted line in Figure 7. Extend a ray from s through u, until it first exits the polygon, the dashed line in the figure. This ray partitions the polygon into two subpolygons, one containing σ and the other containing τ. Consider an arbitrary minimum link path from x to τ; it must cross the extension of $\overline{su}$. Let x' denote the point of crossing. Observe that the path from x to x' has at least two links, whereas we can join x' to s by single link. Thus, the link $\overline{sx'}$ together with a minimum link path between x' and τ gives a minimum link path from s to τ. By a similar transformation on the other end, we obtain a minimum link path between σ and τ whose endpoints are s and t. □

Lemma 4.3 *[Case (Q3)] If there exists an inflection edge $\overline{uv}$ in a geodesic path between a pair of endpoints of σ and τ, then $d_L(\sigma,\tau) = d_L(u,\tau) + d_L(v,\sigma) - 1$, where u is the vertex closer to σ.*

Proof. There are two cases to be considered, depending on whether the hourglass formed by the four paths between σ and τ is open or closed. If the hourglass is open, then

$$d_L(u,\tau) + d_L(v,\sigma) - 1 = 1 + 1 - 1 = 1 = d_L(\sigma,\tau).$$

If the hourglass is closed, we extend the inflection edge in both directions until it first exits the polygon, as shown by the dashed line in Figure 8. This extension partitions the polygon into two subpolygons. There are three cases:

(1) The extension crosses both σ and τ. In this case the hourglass is open, contrary to our assumption.

(2) The extension crosses neither σ nor τ. In this case, the geodesic path between any point in σ to any point in τ contains this same inflection edge. Thus there exists a minimum link path between the two points, one of whose links contains the inflection edge. This implies that there exists a minimum link path between σ and τ containing this inflection edge $\overline{uv}$. The number of links in such a path is clearly given by the claimed equation.

(3) The extension intersects one but not both the segments. Figure 8 illustrates the case where it intersects τ but not σ. In this case $d_L(v,\sigma)$ is equal to $d_L(\tau,\sigma)$, because we could take the first link on the path from v to σ to contain $\overline{vu}$, and this link can be extended to cross τ. We get,

$$d_L(u,\tau) + d_L(v,\sigma) - 1 = 1 + d_L(\tau,\sigma) - 1 = d_L(\sigma,\tau).$$

□

We summarize the main result of this section in the following theorem.

Theorem 4.4 *Given a simple polygon of n vertices, we can preprocess it in $O(n^3)$ time and space so that, for any two line segment σ and τ, we can find their link distance $d_L(\sigma,\tau)$ in $O(\log n)$ time.*

Once again, a less expensive preprocessing can be used to approximate the link distance.

Theorem 4.5 *In $O(n^2)$ time, a data structure of size $O(n^2)$ can be built to support $O(\log n)$ time queries that report an approximate link distance, $\tilde{d}(\sigma,\tau)$, between two line segments σ and τ, where $d_L(\sigma,\tau) \le \tilde{d}(\sigma,\tau) \le d_L(\sigma,\tau)+1$.*

Proof. The query algorithm is the same as the one given above, except that we now use the approximation algorithm for the two-point link distance query in Step (Q2.2). The link distance in Step (Q3) can be computed exactly in $O(n^2)$ time and space preprocessing, since we require only the window partitions based on each of the vertices. □

Remark: Alternatively, we can use the method of [19] to get within 2 of the link distance, using only $O(n)$ preprocessing time and space.

4.2 Queries between convex polygons

Let S and T be two convex polygons, with a total of k vertices. We show how to compute their link distance in $O(\log k \log n)$ time, using the preprocessing of Section 3.1. We assume that S and T are known to lie inside the polygon P; otherwise, it takes $O(\min\{k+n, k\log n\})$ time to test this.

Theorem 4.6 *In $O(n^3)$ time, a data structure of size $O(n^3)$ can be built to support $O(\log k \log n)$ time link distance queries between any two convex polygons of size k that lie within a simple polygon of size n. Alternatively, we can achieve $O(\log n)$ query time, with an error of at most two links in the reported answer.*

Proof. Our idea is to replace S and T by line segments σ and τ, respectively, such that $d_L(S,T) = d_L(\sigma,\tau)$. Computing the link distance between the segments σ and τ takes $O(\log n)$ time, by the result of the previous section. The bottleneck in the algorithm is the determination of the segments σ and τ.

To find the suitable line segment replacements, consider all (euclidean) shortest paths between points of S and T. There are two extreme paths at each end, the most clockwise and the most counterclockwise ones. Let

s and s' be the two vertices corresponding to these extreme paths in S, and let t and t' be the vertices in T. The two extreme paths are then $\pi_G(s,t')$ and $\pi_G(s',t)$. To visualize these paths, imagine surrounding S and T with a rubber band inside P. The resulting shape is called the relative convex hull of $S \cup T$. It is formed by four pieces: (1) the (ccw) boundary of S between the vertices s and s', (2) the (ccw) boundary of T between t and t', (3) the shortest path $\pi_G(s,t')$, and (4) the shortest path $\pi_G(s',t)$. The paths $\pi_G(s,t')$ and $\pi_G(s',t)$ are called *geodesic outer tangents* of S and T. In our algorithm, we replace S and T by segments $\sigma = \overline{ss'}$ and $\tau = \overline{tt'}$, respectively. We prove in the following that $d_L(S,T) = d_L(\sigma,\tau)$.

First, suppose that the hourglass corresponding to the set of all shortest paths between S and T is open. In this case, it is easy to see that $d_L(S,T) = d_L(\sigma,\tau) = 1$, and we are done. So assume in the following that the relative convex hull forms a closed hourglass. We now have two funnels, one on each end, containing S and T, respectively. Let a and b denote the apexes of these two funnels.

Now imagine that S is a source of light, and consider the region in P that is illuminated by it. Under our assumption that the hourglass between S and T is closed, T lies completely inside the dark region, and hence there is a unique window $\overline{uv}$ in the visibility polygon of S that separates S from T. The endpoint u, which is closer to S, coincides with the apex a. By the observation that the extension of $\overline{uv}$ overlaps with one of the funnel edges incident to a, it easily follows that the extension intersects the segment $\overline{ss'}$. In fact, if $\overline{uv}$ is a pinned edge, then its extension intesects the interior of the segment $\overline{ss'}$; otherwise, the extension is tangent to S and the point of tangency coincides with s'. Thus, there exists a minimum link path between S and T whose first edge overlaps with the extension of $\overline{uv}$. A similar argument works for the funnel on the other end. In conclusion, there is a minimum link path whose first and last edges intersect the segments $\sigma = \overline{ss'}$ and $\tau = \overline{tt'}$, respectively.

We now describe how to find s, s', t, t' in time $O(\log k \log n)$. We mimic the algorithm for computing common tangents between pairs of convex polygons, with a total of k vertices, in time $O(\log k)$, (see [14]). Each step of the algorithm picks a pair of vertices $x \in S$ and $y \in T$, and checks in constant time whether the segment $\overline{xy}$ is reflex, supporting, or concave with respect to each polygon. Depending upon this classification, there are nine possible cases, and in each case a portion of one or both of the convex polygons is eliminated. In our algorithm we test the polygons against the geodesic path $\pi_G(x,y)$, instead of the line segment. The same nine cases arise, and in each case a portion of one or both of the convex polygons is eliminated. After $O(\log k)$ steps the relative outer tangents are found. Since each shortest path computation takes $O(\log n)$ time, the total running time is $O(\log k \log n)$.

We conclude the proof by showing how to achieve an $O(\log n)$ query time, with an error of at most two links. Simply replace polygons S and T, by arbitrary vertices $s \in S$ and $t \in T$, and apply the two point query, to find $d_L(s,t)$. Clearly, $d_L(S,T) + 2 \le d_L(s,t)$, because we could take $\pi_L(S,T)$ and extend it to s and t using at most two more links. □

As in the case of points and lines, we can also obtain approximate link distance with cheaper preprocessing.

Theorem 4.7 *In $O(n^2)$ time, a data structure of size $O(n^2)$ can be built to support $O(\log k \log n)$ time queries that report an approximate link distance, $\tilde{d}(S,T)$, between two convex polygons S and T, where $d_L(S,T) \le \tilde{d}(S,T) \le d_L(S,T) + 1$. Alternatively, we can achieve $O(\log n)$ query time, with a maximum error of 3 links.*

4.3 Queries between simple polygons

If S and T are simple, though not necessarily convex, polygons, then we can only answer the link distance query in time $O(k \log^2 n)$. We claim that a query time that is polylogarithmic in both n and k is not possible. Indeed, we show that $\Omega(k)$ is a lower bound on the query time. Refer to Figure 9 for our lower bound construction. Let S be a point, and T be a polygon that is convex, except for the four consecutive segments on its boundary that are used to form a hook. The link distance between S and T is 1 if the hook is present, and it is 2 otherwise. A query algorithm that solves the problems in less than k steps cannot examine the entire boundary of T, and hence it cannot detect the presence of the hook, thus leading to a wrong answer. In the full paper, we give an $O(k \log^2 n)$ time algorithm for computing the link distance between S and T.

Theorem 4.8 *In $O(n^3)$ time, a data structure of size $O(n^3)$ can be built to support $O(k \log^2 n)$ time link distance queries between any two simple polygons of size k that lie within a simple polygon of size n. A lower bound on the query time is $\Omega(k + \log n)$.*

5 Simultaneous Minimization of Link and Geodesic

In many applications it is desirable to minimize both the length of a path and the number of its links. We show how to obtain a path whose length is at most $\sqrt{2}$ times the shortest path and that has at most twice the number of links as the minimum link path.

Lemma 5.1 *Given a simple polygon, and two points s and t in the polygon, there exists a path with at most*

$2d_L(s,t)$ links whose length is at most $\sqrt{2}d_G(s,t)$. After performing the $O(\log n)$-time link distance and geodesic distance queries to obtain $d_L(s,t)$ and $d_G(s,t)$, this path can be reported in time $O(d_L)$.

Proof. (Sketch) Consider the greedy minimum link path $\pi_L(s,t)$ between s and t. Observe that each link of $\pi_L(s,t)$ touches some vertex of P. A simple geometric argument shows that if the path $\pi_L(s,t)$ turns less than 90 degrees at each bend, then its total length is at most $\sqrt{2}d_G(s,t)$. (In a polygonal chain $\{u,v,w\}$, the turning angle at v is defined to be the absolute value of the angle from the directed line uv to the directed line vw.) For the bends with turning angle greater than 90 degrees, we add an extra link such that the new turn angles are greater than 90. Refer to Figure 10. This construction at most doubles the number of links, but makes the total length at most $\sqrt{2}$ times the optimal. □

In general polygons with holes, a similar approximation of both the link and Euclidean metrics is not possible. In fact, it is easy to construct a polygon with just one hole so that either a path has 2 links but very large euclidean length or $\Omega(n)$ links but very short total length.

6 Conclusion

In this paper, we have proposed a data structure for answering link distance queries between two objects inside a simple polygon. The objects can be points, segments, or polygons. Although we achieve optimal query time, we believe that the preprocessing time and space bounds can be improved. It seems quite plausible that the preprocessing bounds can be improved to $O(nE)$, where E is the size of the visibility graph of P. E can be $\Omega(n^2)$ in the worst case, but in many cases it is much smaller.

Another possible direction is to find better approximation algorithms. We can compute the link distance with an error of 1, with $O(n^2)$ time and space preprocessing. On the other hand, a simple $O(n)$ time and space preprocessing allows the link distance approximation with an error of 4. Can we achieve the approximation error of 1 with a linear-size data structure?

The problem of 2-point link distance query remains open for polygon with holes. The only result in that direction is an $O(n^2 \log^2 n)$ time algorithm for computing the link distance between two fixed points [11].

The link distance problems in three dimensions remain largely unexplored. It is likely that finding exact link distance is quite hard, in which case approximation schemes should be investigated.

References

[1] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri, and C.K. Yap, "Finding minimal convex nested polygons", *Information and Computation*, **83** (1989), pp. 98–110.

[2] M. de Berg, "On Rectilinear Link Distance", *Computational Geometry: Theory and Applications*, **1** (1991), pp. 13–34.

[3] B. Chazelle, "Efficient Polygon Triangulation", Technical Report CS-TR-249-90, Princeton University, 1990.

[4] Dobkin, Personal communication with the third author, 1990.

[5] L.J. Guibas and J. Hershberger, "Optimal Shortest Path Queries in a Simple Polygon", *Proc. Third Annual ACM Symposium on Computational Geometry*, Waterloo, Ontario, 1987, pp. 50–63.

[6] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. "Approximating Polygons and Subdivisions with Minimum Link Paths," *Second Annual International Symposium on Algorithms*, December 16-18, 1991, Taipei, Taiwan, R.O.C.

[7] J. Hershberger, "Finding the Visibility Graph of a Simple Polygon in Time Proportional to its Size", *Proc. Third Annual ACM Symposium on Computational Geometry*, Waterloo, Ontario, 1987, pp. 11–20.

[8] W. Lenhart, R. Pollack, J. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides and C. Yap, "Computing the link center of a simple polygon," *Discrete and Computational Geometry*, **3**(3), pp. 281-293, 1988.

[9] Y. Ke, "An efficient algorithm for link distance problems," *Proc. 5th Annual ACM Symposium on Computational Geometry*, June 5-7, 1989, pp. 69–78.

[10] A. A. Melkman, "On-Line Construction of the Convex Hull of a Simple Polygon", *Information Processing Letters*, **25** (1987), pp. 11–12.

[11] J.S.B. Mitchell, G. Rote, and G. Woeginger, "Minimum-Link Paths Among Obstacles in the Plane", *Proc. 6th Annual ACM Symposium on Computational Geometry*, Berkeley, CA, June 6-8, 1990, pp. 63–72. To appear, *Algorithmica*.

[12] B. K. Natarajan, "On comparing and compressing piecewise linear curves," Technical Report, Hewlett Packard, 1991.

[13] W. P. Niedringhaus, "Scheduling with queueing: the space factory problem," Technical Report, Princeton University, 1979.

[14] M. H. Overmars and J. van Leeuwen, "Maintenance of Configurations in the Plane," *J. Comput. and Syst. Sci.*, **23**, 1981, pp. 166-204.

[15] J. H. Reif and J. A. Storer, "Minimizing turns for discrete movement in the interior of a polygon," *IEEE J. on Robotics and Automation*, 1987, pp. 182-193.

[16] J. A. Storer, "On minimal node-cost planar embeddings," *Networks*, 1984, pp. 181-212.

[17] S. Suri, "A Linear Time Algorithm for Minimum Link Paths inside a Simple Polygon", *Computer Vision, Graphics, and Image Processing,* **35** (1986), pp. 99–110.

[18] S. Suri, "Minimum Link Paths in Polygons and Related Problems", Ph.D. Thesis, Department of Computer Science, The Johns Hopkins University, August, 1987.

[19] S. Suri, "On Some Link Distance Problems in a Simple Polygon", *IEEE Transactions on Robotics and Automation,* **6** (February 1990), pp. 108–113.

[20] S. Suri and J. O'Rourke, "Finding minimal nested polygons", *Proc. 23rd Ann. Allerton Conf. Comm. Control Comput.*, 1985, pp. 470–479.

[21] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," *SIAM J. of Computing,* 1986.

[22] C. A. Wang and E. Chan. "Finding the minimum visible vertex distance between two non-intersecting simple polygons," *Proc. of 2nd Symposium on Computational Geometry,* pp. 34-42, 1986.

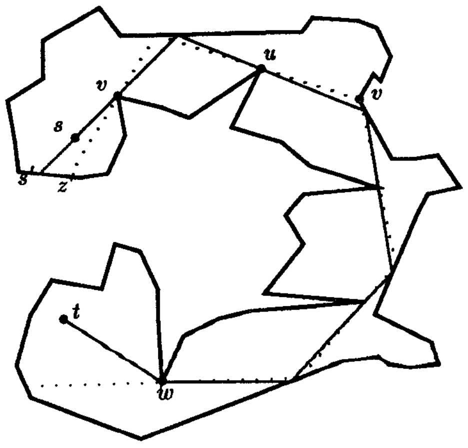

Figure 2: The combinatorial type of the window partition changing at edge $\overline{uv}$.

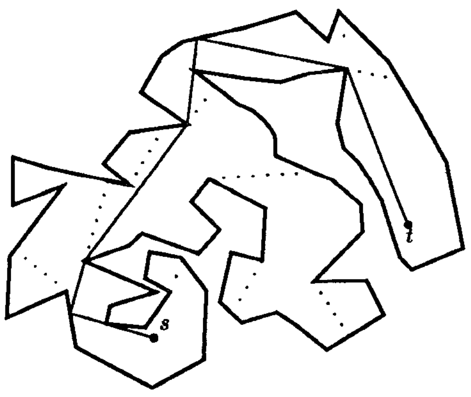

Figure 1: A window partition $WP(s)$ and a corresponding minimum-link path to t.

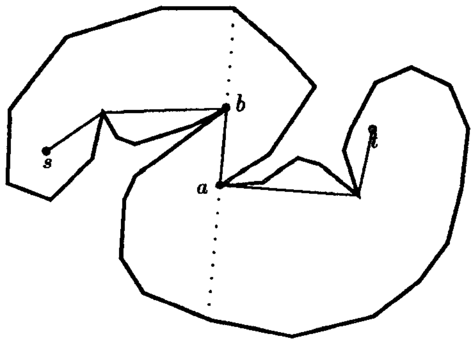

Figure 3: Case (Q3): The path $\pi_G(s,t)$ has an inflection edge $\overline{ab}$.

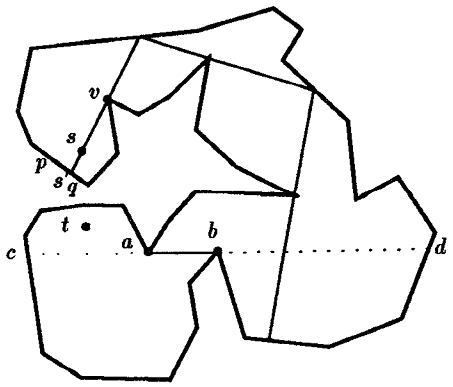

Figure 4: Case (Q4.1): There is a pinned edge $\overline{ab}$ in the greedy path from $\overline{pq}$ to t.

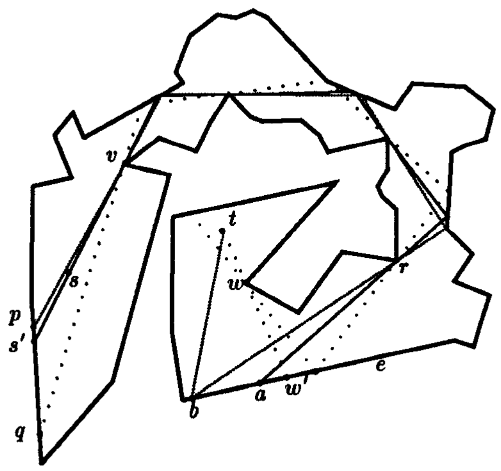

Figure 5: Case (Q4.2): There is no pinned edge in the greedy path from $\overline{pq}$ to t.

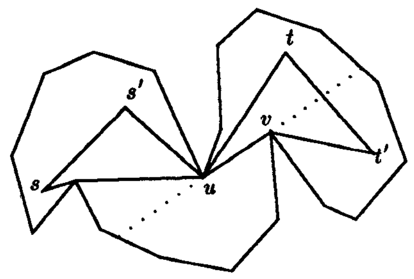

Figure 8: Case (Q3): An inflection edge $\overline{uv}$ in one of $\pi_G(s,t), \pi_G(s,t'), \pi_G(s',t), \pi_G(s',t')$

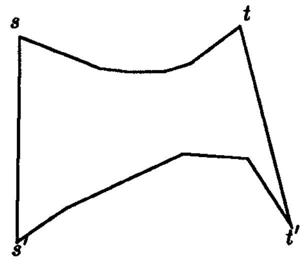

Figure 6: Case (Q2.1): No inflection edges — open hourglass

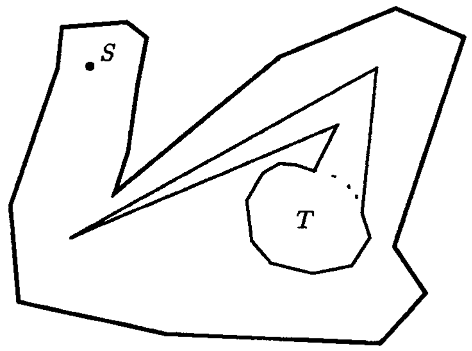

Figure 9: A lower bound on the query time

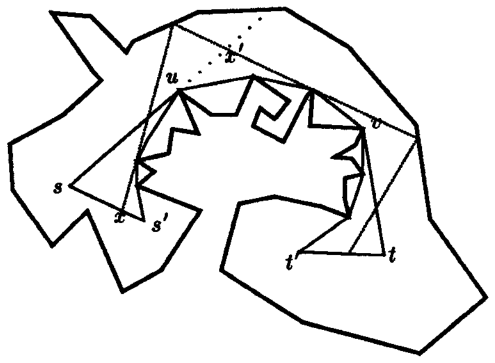

Figure 7: Case (Q2.2): No inflection edges — non-disjoint paths

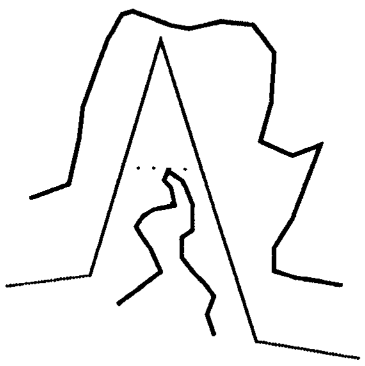

Figure 10: Modifying a bend where the turn angle is greater than 90 degrees.